JASPERREPORTS - REPORT DATASOURCES

Datasources are structured data container. While generating the report, Jasper report engine obtains data from datasources. Data can be obtained from databases, XML files, arrays of objects and collection of objects. We saw in the chapter Filling Reports, the fillReportXXX () method expects to receive a data source of the report that it has to fill, in the form of **net.sf.jasperreports.engine.JRDataSource** object or a **java.sql.Connection** (when the report data is found in a relational database).

The JRDataSource interface has only two methods, which should be implemented:

1. *public boolean next() throws JRException;*

   At the report filling time, this method is called on the data source object by the reporting engine when iterating through the data.

2. *public Object getFieldValue(JRField jrField) throws JRException;*

   This method provides the value for each report field in the current data source record.

The only way to retrieve data from the data source is by using the report fields. There are several default implementations of the JRDataSource interface, depending on the way the records in the data source are acquired.

Datasource Implementations

The table below summarizes the datasources and their implementation classes.

Datasource	Implementation Class
JDBC	net.sf.jasperreports.engine.JRResultSetDataSource
JavaBean	net.sf.jasperreports.engine.data.JRBeanCollectionDataSource, net.sf.jasperreports.engine.data.JRBeanArrayDataSource
Map-based	net.sf.jasperreports.engine.data.JRMapArrayDataSource, net.sf.jasperreports.engine.data.JRMapCollectionDataSource
TableModel	net.sf.jasperreports.engine.data.JRTableModelDataSource
XML	net.sf.jasperreports.engine.data.JRXmlDataSource
CSV	net.sf.jasperreports.engine.data.JRCsvDataSource
XLS	net.sf.jasperreports.engine.data.JRXlsDataSource
Empty	net.sf.jasperreports.engine.JREmptyDataSource

JDBC data sources

Class **JRResultSetDataSource** craps a *java.sql.ResultSet* object. This is the most commonly used data source implementations when report data are extracted from a relational database. If a *java.sql.Connection* is passed to the engine instead, it executes first the related query and stores the returned *java.sql.ResultSet* object in a JRResultSetDataSource instance.

JavaBean data sources

Classes **JRBeanArrayDataSource** and **JRBeanCollectionDataSource** represent implementations that can wrap arrays or collections respectively of JavaBean objects. Each object inside the array or the collection will be seen as one record in this type of data source. The mapping between a particular JavaBean property and the corresponding report field is made by naming conventions. The name of the report field must be the same as the name of the JavaBean property as specified by the JavaBeans specifications.

> *In all the examples in this tutorial we have used JRBeanCollectionDataSource.*

Map-based data sources

The implementation classes **JRMapArrayDataSource** and **JRMapCollectionDataSource** are useful if the parent application already stores the reporting data available in-memory as *java.util.Map objects.* Each Map object in the wrapped array or collection is considered a virtual record in the data source, and the value of each report field is extracted from the map using the report field name as the key.

TableModel data sources

In many client-side applications, data is displayed in tabular format. A common requirement in many applications is to allow the user to print this tabular format as a report. Implementation class **JRTableModelDataSource** makes the task of generating reports from tabular format trivial for Swing applications. This class wraps a javax.swing.table.TableModel object. Columns in the wrapped TableModel object can be accessed either by their names or by their 0-based indexes.

XML data sources

Class **JRXmlDataSource** is a data source implementation based on DOM, which uses XPath expressions to select data from the XML document. Records in the XML data source are represented by node elements selected through the XPath expression. Field values are retrieved from each record using the XPath expression provided by the field description (<fieldDescription> element in JRXML).

> *XPath is a language used to navigate through an XML document's attributes and elements. More information about XPath can be found at http://www.w3.org/TR/xpath.*

CSV data sources

JRCsvDataSource represents an implementation for data sources which retrieve their data from structured text files; usually CSVs. Field values are retrieved using their column index.

XLS data sources

JRXlsDataSource represents an implementation for data sources which retrieve their data from Excel documents. Report-field mapping for this data source implementation is also based on the field column index.

Empty data sources

The class **JREmptyDataSource**, simulates a data source with a given number of virtual empty records inside. It is used

by the UI tools to offer basic report preview functionality, or in special report templates, or for testing and debugging purposes.

Rewindable Data Sources

The **net.sf.jasperreports.engine.JRRewindableDataSource** extends the basic *JRDataSource* interface. It adds only one method, called moveFirst (), to the interface. This method is intended to move the cursor to the first element in the datasource.

Rewindable data sources are useful when working with subreports placed inside a band that is not allowed to split due to the isSplitAllowed="false" setting and there is not enough space on the current page for the subreport to be rendered.

All the above data source implementations are rewindable except for the **JRResultSetDataSource**, as it does not support moving the record pointer back. This poses a problem only if this data source is used to manually wrap a java.sql.ResultSet before passing it to the subreport. There is no problem, if the SQL query resides in the subreport template, as the engine will execute it again when restarting the subreport on the next page.

Data Source Providers

The JasperReports library has an interface **net.sf.jasperreports.engine.JRDataSourceProvider**. This helps in creating and disposing of data source objects. When creating a report template using GUI tools, a special tool for customizing the report's data source is needed. JRDataSourceProvider is the standard way to plug custom data sources into a design tool. A custom implementation of this interface should implement the following methods that allow creating and disposing of data source objects and also methods for listing the available report fields inside the data source if possible:

```
public boolean supportsGetFieldsOperation();

public JRField[] getFields(JasperReport report)
    throws JRException, UnsupportedOperationException;

public JRDataSource create(JasperReport report) throws JRException;

public void dispose(JRDataSource dataSource) throws JRException;
```